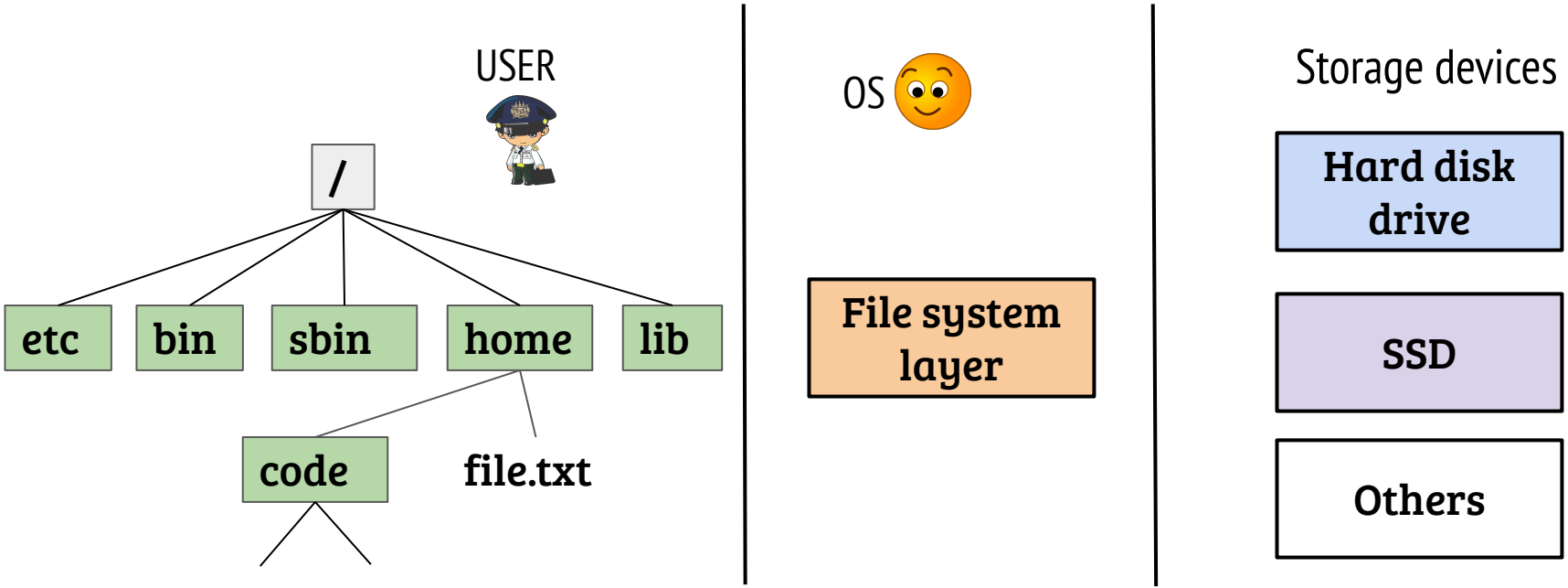
CS330: Operating Systems

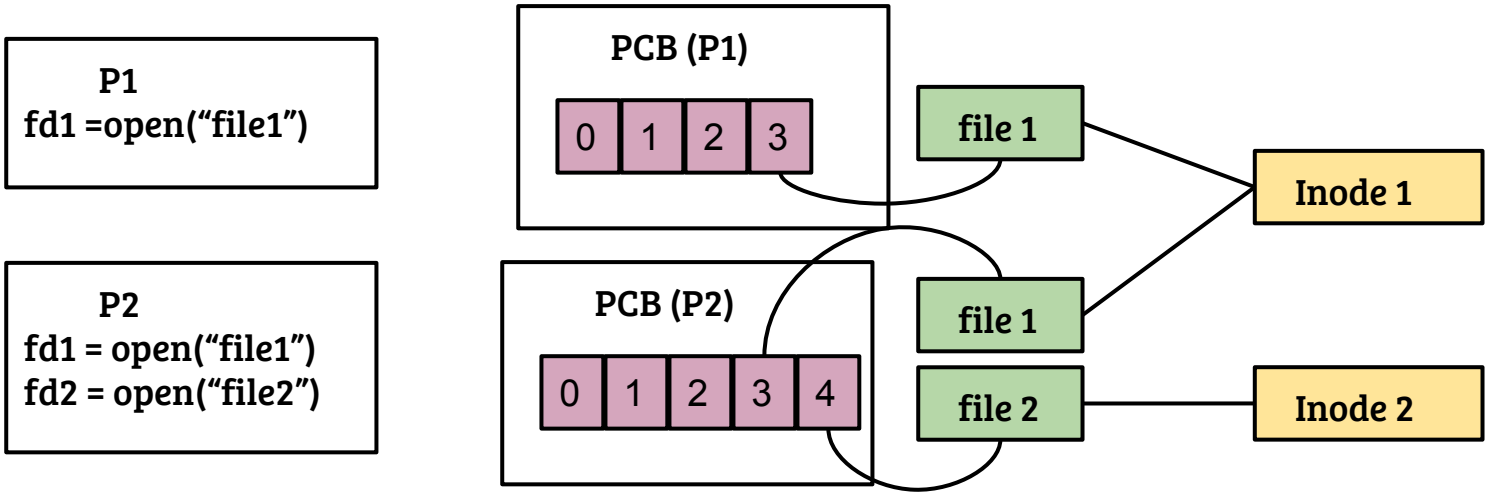
Filesystem

Recap: file system



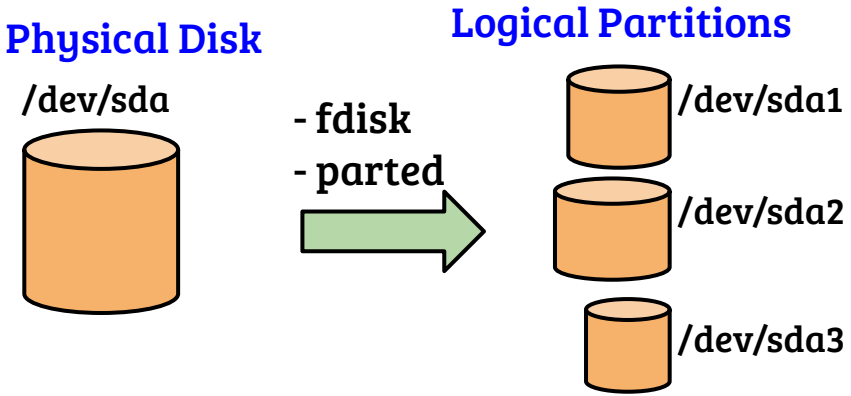
- File system is an important OS subsystem
  - Provides abstractions like files and directories
  - Hides the complexity of underlying storage devices

Recap: Process view of file



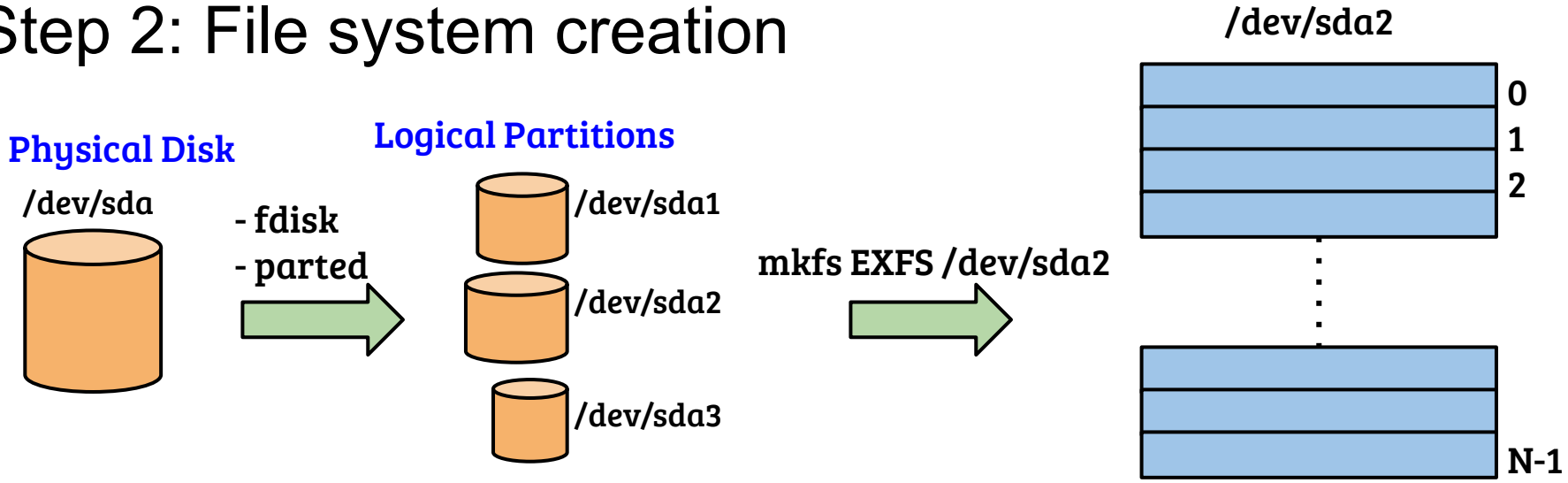
- Per-process file descriptor table with pointer to a "file" object
- file object → inode (in-memory) is many-to-one
- How is the inode maintained in a persistent manner? How to access data at different offsets of a file? How directory structure is maintained?

Step-1: Disk device partitioning



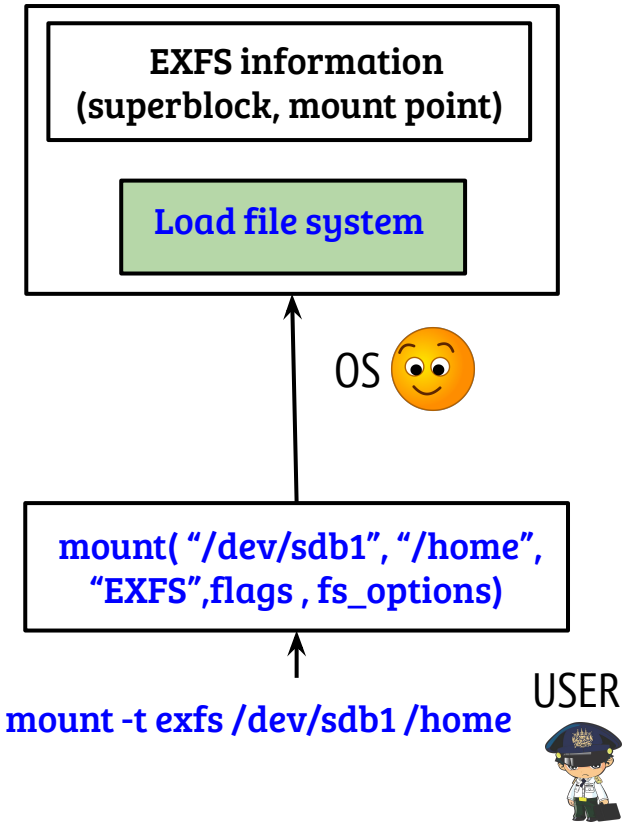
- Partition information is stored in the boot sector of the disk
- Creation of partition is the first step
  - It does not create a file system
- A file system is created on a partition to manage the physical device and present the logical view
- All file systems provide utilities to initialize file system on the partition (e.g., MKFS)

Step 2: File system creation



- MKFS creates initial structures in the logical partition
  - Creates the entry point to the filesystem (known as the super block)
  - At this point the file system is ready to be mounted

Step 3: File system mounting



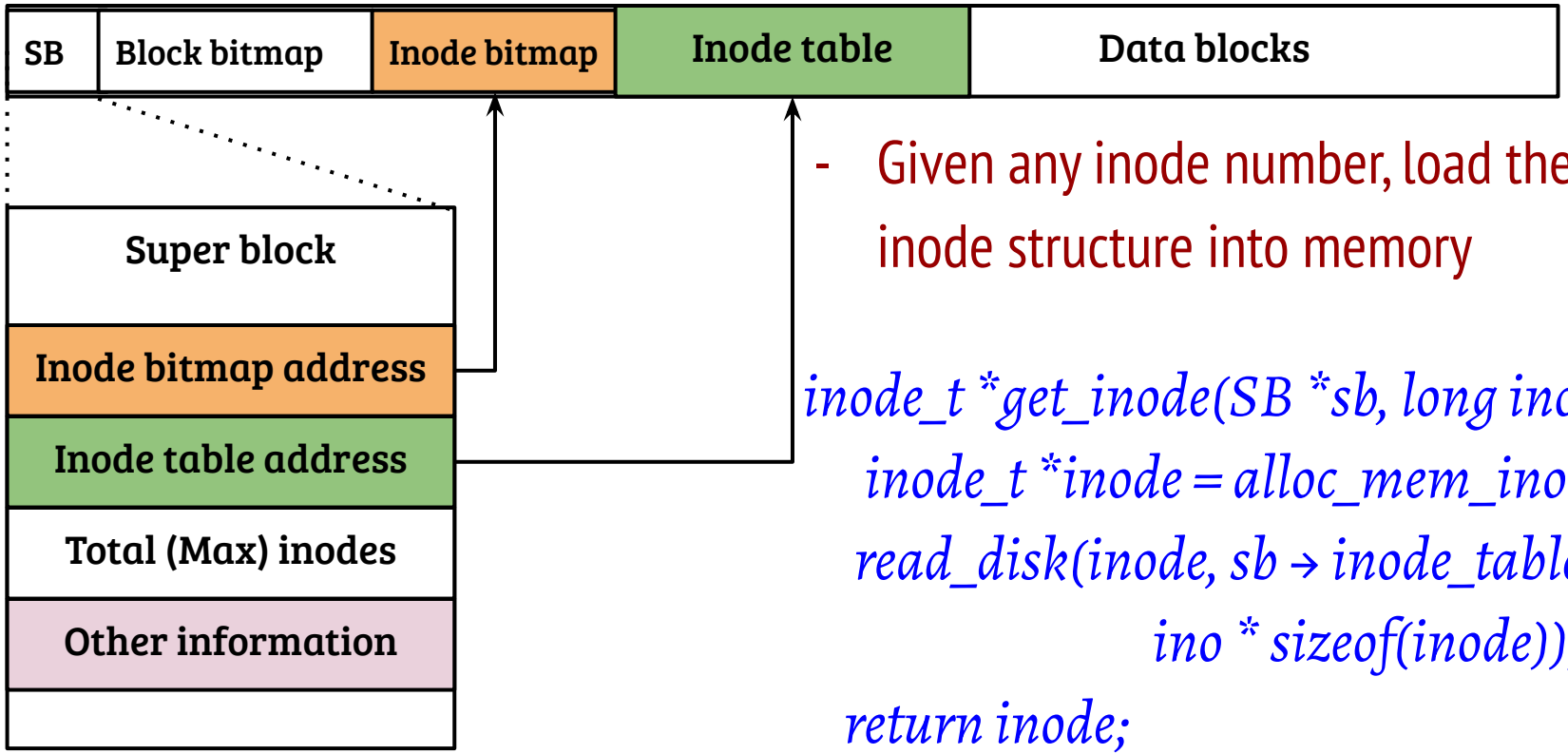
- *mount( )* associates a superblock with the file system mount point
- Example: The OS will use the superblock associated with the mount point "/home" to reach any file/dir under "/home"
- Superblock is a copy of the on-disk superblock along with other information

Structure of an example superblock

```
struct superblock{
    u16 block_size;
    u64 num_blocks;
    u64 last_mount_time;
    u64 root_inode_num;
    u64 max_inodes;
    disk_off_t inode_table;
    disk_off_t blk_usage_bitmap;
     …
};
```

- Superblock contains information regarding the device and the file system organization in the disk
- Pointers to different metadata related to the file system are also maintained by the superblock
  - Ex: List of free blocks is required before adding data to a new file/directory

File system organization



- Given any inode number, load the inode structure into memory

```
inode_t *get_inode(SB *sb, long ino){
    inode_t *inode = alloc_mem_inode( );
    read_disk(inode, sb → inode_table +
                        ino * sizeof(inode));
    return inode;
}
```

File system organization

- File system is mounted, the inode number for root of the file system (i.e., the mount point) is known, root inode can be accessed.
- How to search/lookup files/directories under root inode?
- Specifically,
  - How to locate the content in disk?
  - How to keep track of size, permissions etc.?

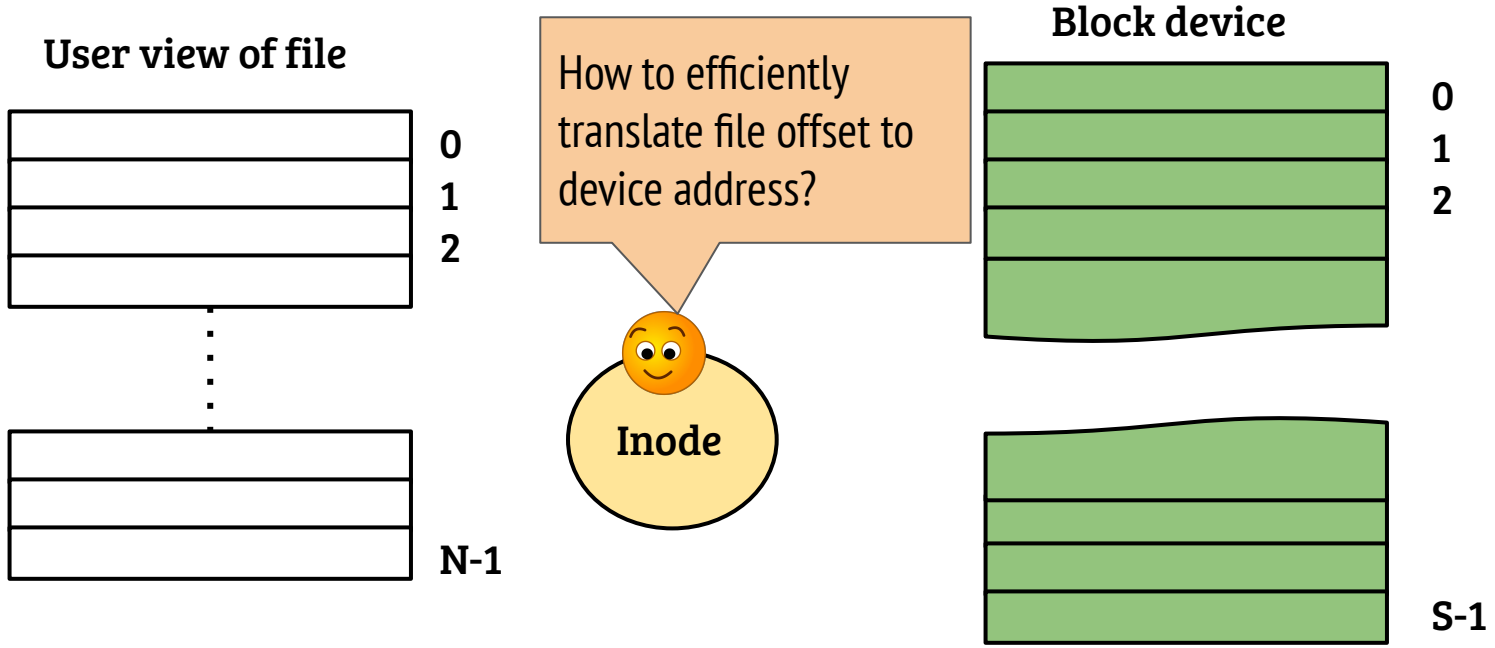
return inode;

}

Inode

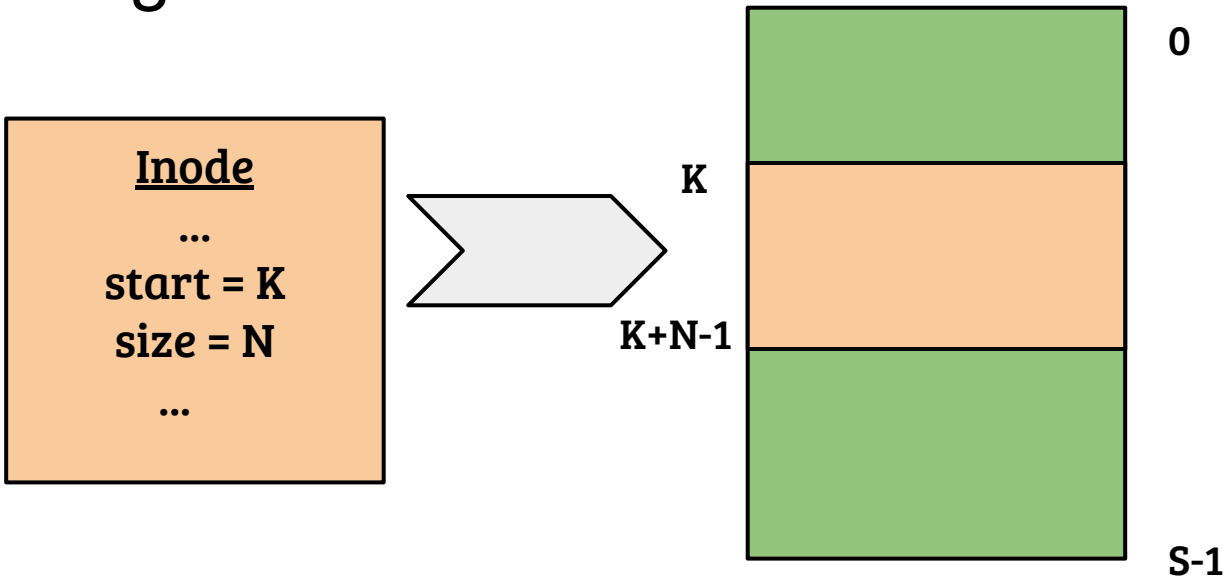
- A on-disk structure containing information regarding files/directories in the unix systems
  - Represented by a unique number in the file system (e.g., in Linux, “ls -i filename” can be used to print the inode)
  - Contains access permissions, access time, file size etc.
  - *Most importantly, inode contains information regarding the file data location on the device*
- Directory inodes also contain information regarding its content, albeit the content is structured (for searching files)

Problem: file offset to disk address mapping



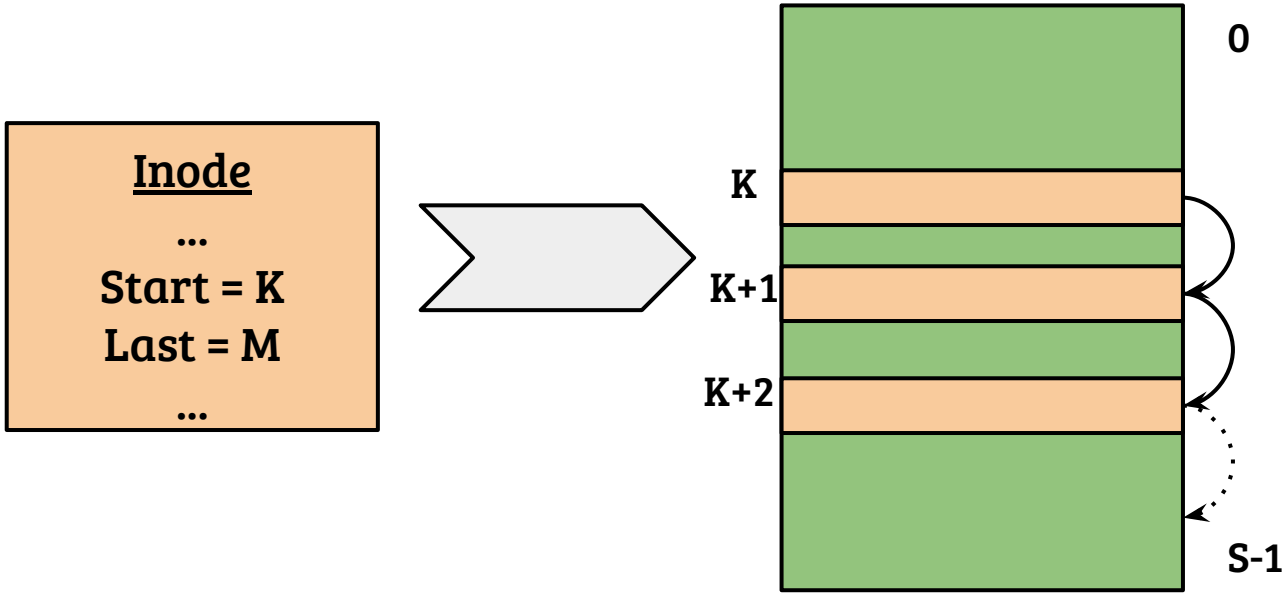
- File size can range from few bytes to gigabytes
- Can be accessed in a sequential or random manner
- How to design the mapping structure?

Contiguous allocation



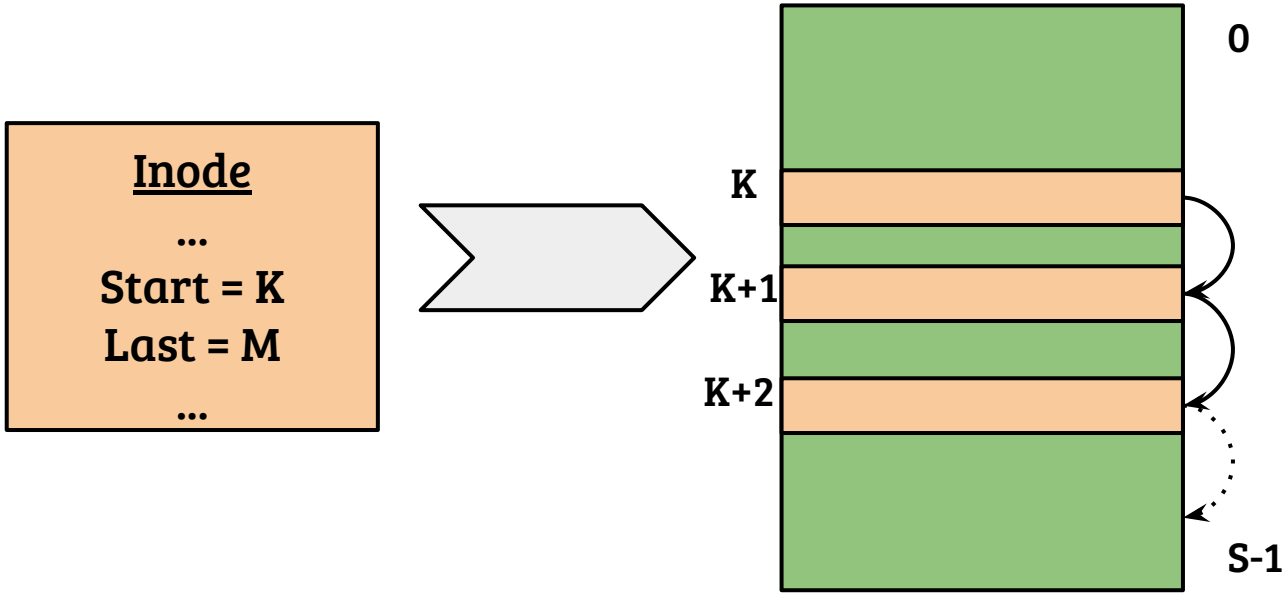
- Works nicely for both sequential and random access
- Append operation is difficult, How to expand files? Require relocation!
- External fragmentation is a concern

Linked allocation



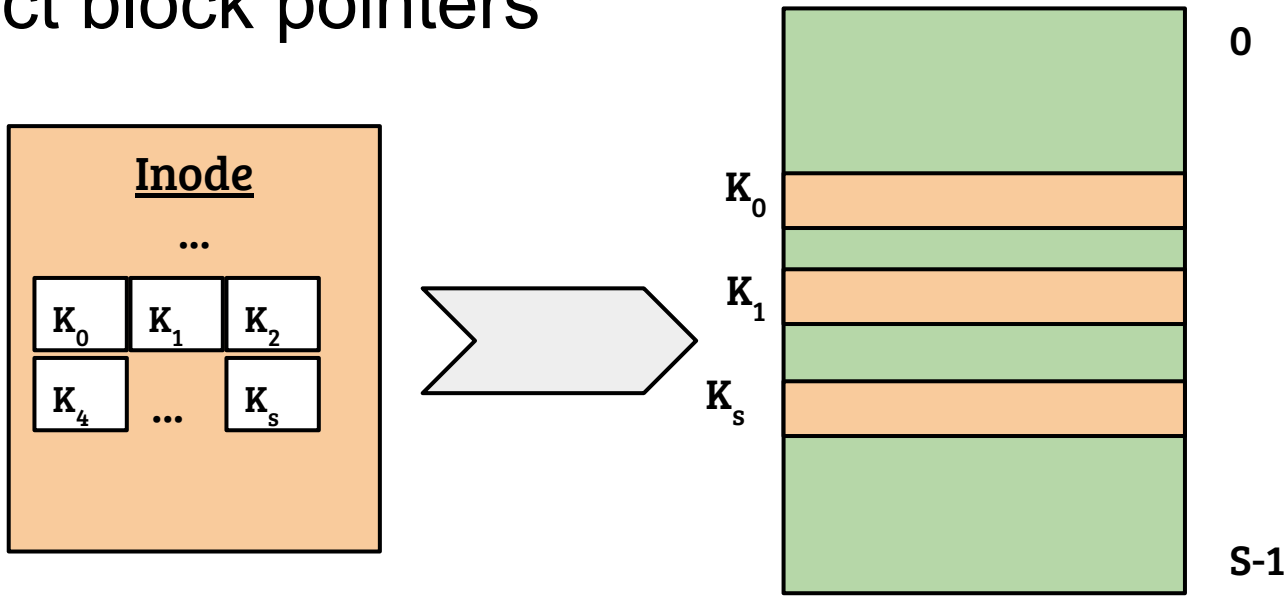
- Every block contains pointer to next block
- Advantage: flexible, easy to grow and shrink, Disadvantage: random access
- Why maintain last block not size?

Linked allocation



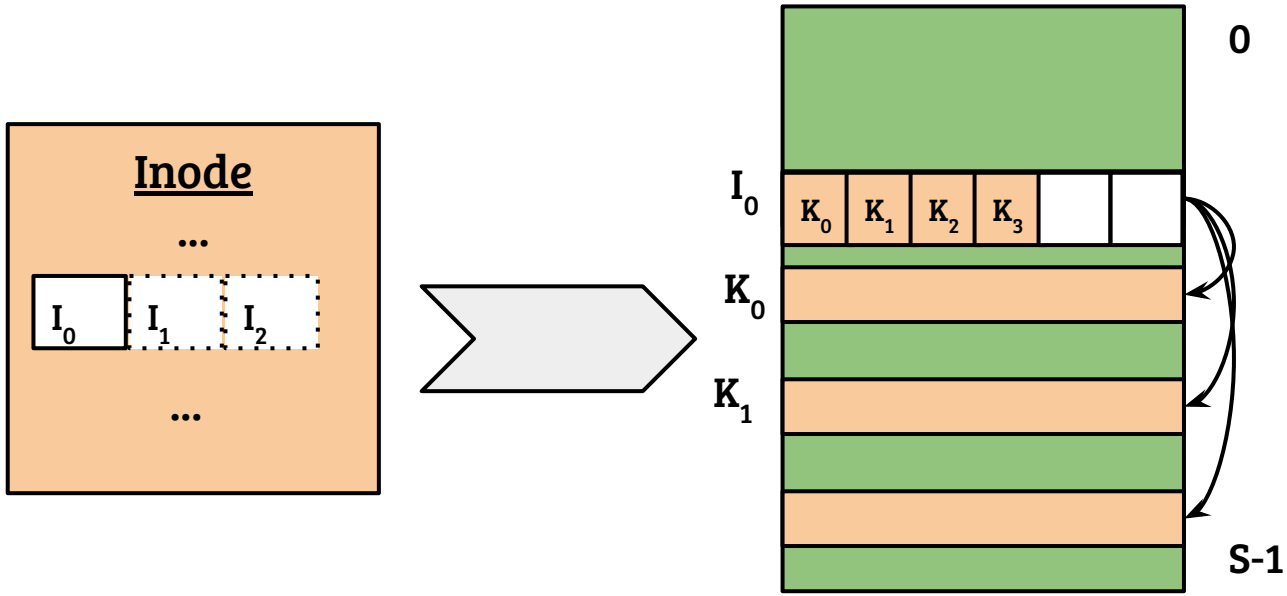
- Every block contains pointer to next block
- Advantage: flexible, easy to grow and shrink, Disadvantage: random access
- Why maintain last block not size? Efficient append operation!

Direct block pointers



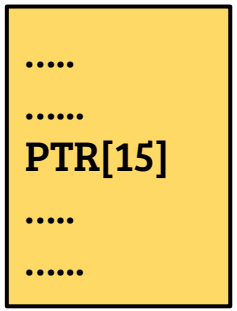
- Inode contains direct pointers to the block
- Flexible: growth, shrink, random access is good
- Can not support files of larger size!

Indirect block pointers



- Inode contains the pointers to a block containing pointers to data blocks
- Advantages: flexible, random access is good
- Disadvantages: Indirect block access overheads (even for small files)

Hybrid block pointers: Ext2 file system



Direct pointers {PTR [0] to PTR [11]}

File block address (0 -11)

Single indirect {PTR [12]}

File block address (12 -1035)

Double indirect {PTR [13]}

File block address (1036 to 1049611)

Triple indirect {PTR [14]}

File block address (?? to ??)

File system organization

- File system is mounted, the inode number for root of the file system (mount point) is known, root inode can be accessed. However,
- How to search/lookup files/directories under root inode?
- Specifically,
  - How to locate the content in disk?
  - Index structures in inode are used to map file offset to disk location
  - How to keep track of size, permissions etc.?
  - Inode is used to maintain these information

Organizing the directory content

Fixed size directory entry

```
struct dir_entry{
        inode_t  inode_num;
        char name[FNAME_MAX];
};
```

- Fixed size directory entry is a simple way to organize directory content
- Advantages: avoid fragmentation, rename
- Disadvantages: space wastage

Organizing the directory content

Fixed size directory entry

```
struct dir_entry{
        inode_t  inode_num;
        char name[FNAME_MAX];
};
```

Variable size directory entry

```
struct dir_entry{
        inode_t  inode_num;
        u8 entry_len;
        char name[name_len];
};
```

- Variable sized directory entries contain length explicitly
- Advantages: less space wastage (compact)
- Disadvantages: inefficient rename, require compaction

File system organization

- File system is mounted, the inode number for root of the file system (mount point) is known, root inode can be accessed. However,
- How to search/lookup files/directories under root inode?
- Read the content of the root inode and search the next level dir/file
- Specifically,
  - How to locate the content in disk?
  - Index structures in inode are used to map file offset to disk location
  - How to keep track of size, permissions etc.?
  - Inode is used to maintain these information